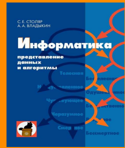
http://rain.ifmo.ru/cat

Алгоритм : <2> оценка эффективности

© С. Е. Столяр, 2010

ses@mail.ifmo.ru



Сравнение алгоритмов

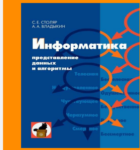
- Получить решение поставленной задачи нередко можно разными способами, привлекая *разные алгоритмы*.

Back

Close

Сравнение алгоритмов

- Получить решение поставленной задачи нередко можно разными способами, привлекая *разные алгоритмы*.
- Естественно стремление выбрать *наиболее эффективный* из конкурирующих алгоритмов.



Сравнение алгоритмов

- Получить решение поставленной задачи нередко можно разными способами, привлекая *разные алгоритмы*.
- Естественно стремление выбрать *наиболее эффективный* из конкурирующих алгоритмов.
- Сравнение алгоритмов правомерно только для *одного и того же* исполнителя и актуально лишь для *массового* применения.

Back
Close

Сравнение алгоритмов

- Получить решение поставленной задачи нередко можно разными способами, привлекая *разные алгоритмы*.
- Естественно стремление выбрать *наиболее эффективный* из конкурирующих алгоритмов.
- Сравнение алгоритмов правомерно только для *одного и того же* исполнителя и актуально лишь для *массового* применения.

Пример 1

$85 \times 85 = ?$ Исполнитель — ученик.

***Алгоритм* 1.** Угадывать, перебирая числа из диапазона $101 \ldots 10\,000$ до «победного конца».

***Алгоритм* 2.** Умножение «в столбик». Требуется *оперативная память* тетрадочного листа.

***Алгоритм* 3.** По формуле $(8 \times (8+1)) \times 100 + 5 \times 5$. Вычисления «в уме».

***Алгоритм* 4.** По «расширенной» таблице умножения 100×100, которую некто предоставил.

Back
Close

Сравнение алгоритмов: критерии

- По расходуемой оперативной памяти — оценивается *ёмкостная сложность*.

  В приведенном выше примере по этому критерию выигрывает алгоритм 1.

Сравнение алгоритмов: критерии

- По расходуемой оперативной памяти — оценивается *ёмкостная сложность*.

  В приведенном выше примере по этому критерию выигрывает алгоритм 1.

- В ряде случаев оцениваются также привлекаемые ресурсы внешней памяти (например, для т. н. *внешних* сортировок).

  В том же примере расход внешней памяти под расширенную таблицу умножения в алгоритме 4 кажется малооправданным.

Back
Close

Сравнение алгоритмов: критерии

- По расходуемой оперативной памяти — оценивается *ёмкостная сложность*.

  В приведенном выше примере по этому критерию выигрывает алгоритм 1.

- В ряде случаев оцениваются также привлекаемые ресурсы внешней памяти (например, для т. н. *внешних* сортировок).

  В том же примере расход внешней памяти под расширенную таблицу умножения в алгоритме 4 кажется малооправданным.

- По времени исполнения *в среднем* — оценка *временно́й сложности*.

  Для того же примера побеждает алгоритм 3.

Back
Close

Сравнение алгоритмов: критерии

- По расходуемой оперативной памяти — оценивается *ёмкостная сложность*.

  В приведенном выше примере по этому критерию выигрывает алгоритм 1.

- В ряде случаев оцениваются также привлекаемые ресурсы внешней памяти (например, для т. н. *внешних* сортировок).

  В том же примере расход внешней памяти под расширенную таблицу умножения в алгоритме 4 кажется малооправданным.

- По времени исполнения *в среднем* — оценка *временно́й сложности*.

  Для того же примера побеждает алгоритм 3.

- По времени исполнения *в худшем случае*.

  Здесь явный аутсайдер — алгоритм 1.

Back
Close

Временна́я сложность

- Оценивать эффективность компьютерного алгоритма следует *до* написания и отладки компьютерной программы.

Временна́я сложность

- Оценивать эффективность компьютерного алгоритма следует *до* написания и отладки компьютерной программы.
- Нередко оценка временно́й эффективности опытным путем, в реальном времени, принципиально невозможна (например, при неоправданно больших затратах машинного времени).

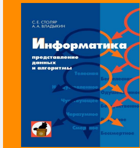
Временна́я сложность

- Оценивать эффективность компьютерного алгоритма следует *до* написания и отладки компьютерной программы.
- Нередко оценка временно́й эффективности опытным путем, в реальном времени, принципиально невозможна (например, при неоправданно больших затратах машинного времени).
- Принято ориентироваться на число *шагов алгоритма*.

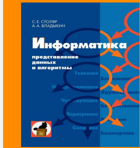
Back
Close

Временна́я сложность

- Оценивать эффективность компьютерного алгоритма следует *до* написания и отладки компьютерной программы.
- Нередко оценка временно́й эффективности опытным путем, в реальном времени, принципиально невозможна (например, при неоправданно больших затратах машинного времени).
- Принято ориентироваться на число *шагов алгоритма*.
- Под шагом редко понимают *машинную операцию* (инструкцию).



Back
Close

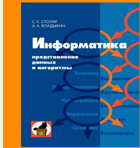


Временна́я сложность

- Оценивать эффективность компьютерного алгоритма следует *до* написания и отладки компьютерной программы.
- Нередко оценка временно́й эффективности опытным путем, в реальном времени, принципиально невозможна (например, при неоправданно больших затратах машинного времени).
- Принято ориентироваться на число *шагов алгоритма*.
- Под шагом редко понимают *машинную операцию* (инструкцию).
- Обычно это инструкция *абстрактного* исполнителя, не требующая более подробного алгоритмического измельчения.

Временна́я сложность

- Оценивать эффективность компьютерного алгоритма следует *до* написания и отладки компьютерной программы.
- Нередко оценка временно́й эффективности опытным путем, в реальном времени, принципиально невозможна (например, при неоправданно больших затратах машинного времени).
- Принято ориентироваться на число *шагов алгоритма*.
- Под шагом редко понимают *машинную операцию* (инструкцию).
- Обычно это инструкция *абстрактного* исполнителя, не требующая более подробного алгоритмического измельчения.
- *Алгоритмическое* время выполнения одного шага либо не должно зависеть от параметров задачи, либо стоимость *укрупненного шага* известна и будет учитываться в общей оценке.



Учет параметров

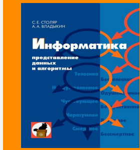
Пример 2

Найти сумму натуральных чисел от 1 до заданного n.

Алгоритм 1. Вычисление по формуле для суммы арифметической прогрессии. В этом случае временна́я сложность не зависит от значения параметра n.

Алгоритм 2. Начав с нуля, накапливаем сумму, добавляя на каждом *шаге* очередное слагаемое.

Для малых значений n выгоднее *алгоритм* 2, в остальных же случаях — *алгоритм* 1.



Back
Close

Учет параметров

Пример 2

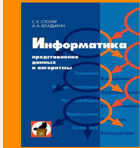
Найти сумму натуральных чисел от 1 до заданного n.

Алгоритм 1. Вычисление по формуле для суммы арифметической прогрессии. В этом случае временна́я сложность не зависит от значения параметра n.

Алгоритм 2. Начав с нуля, накапливаем сумму, добавляя на каждом *шаге* очередное слагаемое.

Для малых значений n выгоднее *алгоритм* 2, в остальных же случаях — *алгоритм* 1.

- Число шагов *алгоритма* 2 есть некоторая функция от количества обрабатываемых элементов — $f(n)$.
- Каждый его шаг включает выполнение *нескольких* операторов программы (или машинных инструкций), но всегда одних и тех же, число их никак не зависит от параметра n.
- Время работы этого алгоритма *линейно* зависит от значения n.



Back
Close

Асимптотическая нотация

Анализируя поведение функций, применяют специальную символику, *O-нотацию*[1](еще одно название: *нотация Бахмана-Ландау*[2]).

- Оценивая скорость роста некой величины, ее сравнивают с какой-нибудь функцией, чье поведение хорошо исследовано.

Back

Close

Асимптотическая нотация

Анализируя поведение функций, применяют специальную символику, *O-нотацию*[1](еще одно название: *нотация Бахмана-Ландау*[2]).

- Оценивая скорость роста некой величины, ее сравнивают с какой-нибудь функцией, чье поведение хорошо исследовано.
- Обозначение $g(n) = O(f(n))$ будем относить к дискретным функциям $f(n)$ натурального n и ко всем функциям $g(n)$, растущим не быстрее $f(n)$.

Асимптотическая нотация

Анализируя поведение функций, применяют специальную символику, *O-нотацию*[1](еще одно название: *нотация Бахмана-Ландау*[2]).

- Оценивая скорость роста некой величины, ее сравнивают с какой-нибудь функцией, чье поведение хорошо исследовано.
- Обозначение $g(n) = O(f(n))$ будем относить к дискретным функциям $f(n)$ натурального n и ко всем функциям $g(n)$, растущим не быстрее $f(n)$.
- Формулировка «растущим не быстрее» означает, что существует такая пара положительных значений M и n_0, что $g(n) \leq Mf(n)$ для $n \geq n_0$.

Back

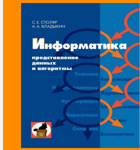
Close

Асимптотическая нотация

Анализируя поведение функций, применяют специальную символику, *O-нотацию*[1](еще одно название: *нотация Бахмана-Ландау*[2]).

- Оценивая скорость роста некой величины, ее сравнивают с какой-нибудь функцией, чье поведение хорошо исследовано.
- Обозначение $g(n) = O(f(n))$ будем относить к дискретным функциям $f(n)$ натурального n и ко всем функциям $g(n)$, растущим не быстрее $f(n)$.
- Формулировка «растущим не быстрее» означает, что существует такая пара положительных значений M и n_0, что $g(n) \leq M f(n)$ для $n \geq n_0$.
- Если же вместе с этим выполняется и $f(n) = O(g(n))$, то говорят, что функция $g(n)$ — «порядка $f(n)$ для больших n».

Back
Close



Асимптотическая нотация

Анализируя поведение функций, применяют специальную символику, *O-нотацию*[1](еще одно название: *нотация Бахмана-Ландау*[2]).

- Оценивая скорость роста некой величины, ее сравнивают с какой-нибудь функцией, чье поведение хорошо исследовано.
- Обозначение $g(n) = O(f(n))$ будем относить к дискретным функциям $f(n)$ натурального n и ко всем функциям $g(n)$, растущим не быстрее $f(n)$.
- Формулировка «растущим не быстрее» означает, что существует такая пара положительных значений M и n_0, что $g(n) \leq Mf(n)$ для $n \geq n_0$.
- Если же вместе с этим выполняется и $f(n) = O(g(n))$, то говорят, что функция $g(n)$ — «порядка $f(n)$ для больших n».
- O-нотация дает верхнюю оценку временно́й сложности алгоритма, его *асимптотическую сложность*.
  $\mathcal{NB}$! Использование констант M и n_0 фактически связано с «большими» значениями аргумента n и мало что дает при его малых значениях.

[1] Читается: *O-большое*.
[2] P. Bachmann, 1894; E. Landau, 1909.

Back
Close

Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

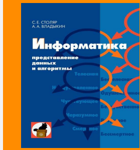
Back

Close

Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

(2) $c \cdot O(f(n)) = O(f(n))$, где c — некоторая константа.



Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

(2) $c \cdot O(f(n)) = O(f(n))$, где c — некоторая константа.

(3) $O(f(n)) + O(f(n)) = O(f(n))$.

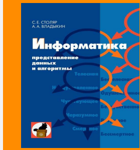
Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

(2) $c \cdot O(f(n)) = O(f(n))$, где c — некоторая константа.

(3) $O(f(n)) + O(f(n)) = O(f(n))$.

(4) $O(O(f(n))) = O(f(n))$.



Back
Close

Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

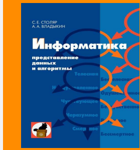
(2) $c \cdot O(f(n)) = O(f(n))$, где c — некоторая константа.

(3) $O(f(n)) + O(f(n)) = O(f(n))$.

(4) $O(O(f(n))) = O(f(n))$.

(5) $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$.

Back
Close



Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

(2) $c \cdot O(f(n)) = O(f(n))$, где c — некоторая константа.

(3) $O(f(n)) + O(f(n)) = O(f(n))$.

(4) $O(O(f(n))) = O(f(n))$.

(5) $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$.

В *Примере 2* (см. слайд *5*) *алгоритм* 2 имеет *линейную сложность*, т .к. его быстродействие, то есть число шагов, согласно свойству (1) есть $O(n)$.

Back
Close

Некоторые свойства O-операций

(1) $f(n) = O(f(n))$.

(2) $c \cdot O(f(n)) = O(f(n))$, где c — некоторая константа.

(3) $O(f(n)) + O(f(n)) = O(f(n))$.

(4) $O(O(f(n))) = O(f(n))$.

(5) $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$.

В *Примере 2* (см. слайд *5*) *алгоритм* 2 имеет *линейную сложность*, т .к. его быстродействие, то есть число шагов, согласно свойству (1) есть $O(n)$.

o-нотация

- Обозначение $g(n) = o(f(n))$ относится к функциям, для которых отношение $g(n)/f(n)$ стремится к 0 при росте n.
- Читается: *o-малое*.

Примеры . . .

Пример 3

Выбрать пару соседей для первой парты в классе из n учеников можно $n(n-1)$ способами.

Трудоемкость перебора всевозможных вариантов оценивается как $O(n^2)$, или *квадратичная*.

Back
Close

Примеры ...

Пример 3

Выбрать пару соседей для первой парты в классе из n учеников можно $n(n-1)$ способами.

Трудоемкость перебора всевозможных вариантов оценивается как $O(n^2)$, или *квадратичная*.

Пример 4

Из n попарно неравных отрезков можно составить $n(n-1)(n-2)$ невырожденных треугольников.

Стоимость перебора всех вариантов — $O(n^3)$, т. е. *кубическая*.

Back

Close

Примеры . . .

Пример 3

Выбрать пару соседей для первой парты в классе из n учеников можно $n(n-1)$ способами.

Трудоемкость перебора всевозможных вариантов оценивается как $O(n^2)$, или *квадратичная*.

Пример 4

Из n попарно неравных отрезков можно составить $n(n-1)(n-2)$ невырожденных треугольников.

Стоимость перебора всех вариантов — $O(n^3)$, т. е. *кубическая*.

Пример 5

В машинном слове записано натуральное число n. Какова трудоемкость операции $\mathrm{odd}(n)$? —

Достаточно проверить младший бит, поэтому алгоритм имеет *константную* сложность, $O(1)$.

Back
Close

. . . Примеры

Пример 6

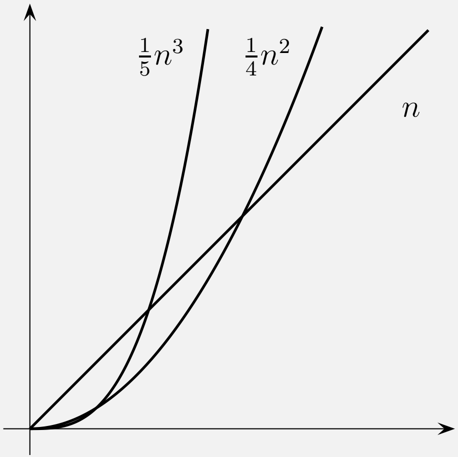
Говорят, что алгоритм имеет *полиномиальную* сложность с оценкой $O(n^k)$, если его эффективность определяется вычислительной сложностью обработки многочлена порядка k. Действительно, согласно свойству (2), старший коэффициент можно опустить.

. . . Примеры

Пример 6

Говорят, что алгоритм имеет *полиномиальную* сложность с оценкой $O(n^k)$, если его эффективность определяется вычислительной сложностью обработки многочлена порядка k. Действительно, согласно свойству (2), старший коэффициент можно опустить.

На графиках видно, что сколь бы ни был мал коэффициент при старшем k-члене и, напротив, велик — при любом другом m-члене $(m < k)$, вклад первого из них в поведение всего многочлена рано или поздно, для «больших» n, становится решающим.



Более сложные задачи

- Существует много задач, алгоритмы решения которых имеют трудоемкость $O(2^n), O(n!), O(n^n)$ и более.

Back
Close

Более сложные задачи

- Существует много задач, алгоритмы решения которых имеют трудоемкость $O(2^n), O(n!), O(n^n)$ и более.
- Такие алгоритмы принято характеризовать как имеющие *экспоненциальную* сложность.

Более сложные задачи

- Существует много задач, алгоритмы решения которых имеют трудоемкость $O(2^n), O(n!), O(n^n)$ и более.
- Такие алгоритмы принято характеризовать как имеющие *экспоненциальную* сложность.
- Предположительно[3], для некоторых классов задач невозможно построить алгоритм, эффективнее экспоненциального.

Более сложные задачи

- Существует много задач, алгоритмы решения которых имеют трудоемкость $O(2^n), O(n!), O(n^n)$ и более.
- Такие алгоритмы принято характеризовать как имеющие *экспоненциальную* сложность.
- Предположительно[3], для некоторых классов задач невозможно построить алгоритм, эффективнее экспоненциального.
- Решая подобные задачи в условиях дефицита вычислительных ресурсов, прибегают к *приближенным алгоритмам*.

Более сложные задачи

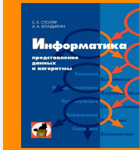
- Существует много задач, алгоритмы решения которых имеют трудоемкость $O(2^n), O(n!), O(n^n)$ и более.
- Такие алгоритмы принято характеризовать как имеющие *экспоненциальную* сложность.
- Предположительно[3], для некоторых классов задач невозможно построить алгоритм, эффективнее экспоненциального.
- Решая подобные задачи в условиях дефицита вычислительных ресурсов, прибегают к *приближенным алгоритмам*.
- Разумеется, приближенный алгоритм позволяет найти лишь *приближенное решение*.

[3]Это пока не доказано теоретически.

Приближенные алгоритмы

Пример 7

- Известна формула длины окружности $S = 2\pi r$, или $\pi = S/(2r)$.
- Заменив окружность правильным n-угольником, получим приближенное значение π_n.



Back
Close

Приближенные алгоритмы

Пример 7

- Известна формула длины окружности $S = 2\pi r$, или $\pi = S/(2r)$.
- Заменив окружность правильным n-угольником, получим приближенное значение π_n.
- *Алгоритм.*
  Последовательно увеличая значение параметра n, можно увеличивать *точность* соответствующих значений π_n.

Back
Close

Приближенные алгоритмы

Пример 7

- Известна формула длины окружности $S = 2\pi r$, или $\pi = S/(2r)$.
- Заменив окружность правильным n-угольником, получим приближенное значение π_n.
- *Алгоритм.*
  Последовательно увеличивая значение параметра n, можно увеличивать *точность* соответствующих значений π_n.

ДЗ Напишите программу, реализующую описанный алгоритм и выводящую значение n, начиная с которого условие $\pi_n > 3.14$ не нарушается.

Вопросы / упражнения / задания

$\boxed{?}$ Можете ли вы сформулировать задачу и алгоритм ее решения, имеющий *логарифмическую* временну́ю сложность?

Вопросы / упражнения / задания

$\boxed{?}$ Можете ли вы сформулировать задачу и алгоритм ее решения, имеющий *логарифмическую* временну́ю сложность?

$\boxed{\text{ДЗ}}$ Сформулируйте задачу и алгоритм ее решения, имеющий *экспоненциальную* сложность.

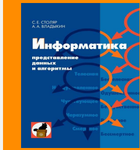
Back
Close

Вопросы / упражнения / задания

? Можете ли вы сформулировать задачу и алгоритм ее решения, имеющий *логарифмическую* временну́ю сложность?

ДЗ Сформулируйте задачу и алгоритм ее решения, имеющий *экспоненциальную* сложность.

ДЗ То же задание для *факториальной* трудоемкости.



Вопросы / упражнения / задания

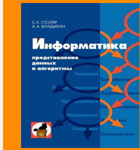
? Можете ли вы сформулировать задачу и алгоритм ее решения, имеющий *логарифмическую* временну́ю сложность?

ДЗ Сформулируйте задачу и алгоритм ее решения, имеющий *экспоненциальную* сложность.

ДЗ То же задание для *факториальной* трудоемкости.

Ретроспектива и перспектива учебного курса

Любой алгоритм нуждается в оценке его ёмкостной и временно́й сложности!



Back
Close

Остались вопросы?

Рекомендуем заглянуть в предлагаемые источники:

- С. Е. Столяр, А. А. Владыкин. *Информатика: Представление данных и алгоритмы.* — СПб.: Невский Диалект; М.: БИНОМ. Лаборатория знаний, 2007. — 382 с.

- *O-нотация* =
  `http://en.wikipedia.org/wiki/Big_O_notation` [ONLINE]
  `http://ru.wikipedia.org/wiki/[<<O>> большое и <<o>> малое]` [ONLINE]